Case Study: Developing a quality assurance test for an electronic throttle body.

Scott Hirsch
Head of MATLAB Product Management
The MathWorks
shirsch@mathworks.com
508-647-7882

Summary

This case study demonstrates the Test and Measurement marketing concept of an "Integrated Measurement System." The approach of our marketing efforts is to demonstrate the value of integrating your test applications with your analysis and simulation. The case study is the design of a quality assurance test for an electronic throttle body. The three primary capabilities demonstrated are: 1) hardware connectivity, 2) test data analysis / algorithm development, and 3) integration of test and simulation.

NOTE: The demo has been modified slightly for release on MATLAB Central. Don't be surprised if some things in this documentation don't match exactly the files you've got!

Table of Contents

Introduction

More information on the throttle body can be found in the January, 2002 issue of MATLAB Digest. There is an article detailing a different application with the same hardware. This article can be found at http://www.mathworks.com/company/digest/jan02/throttle.shtml.

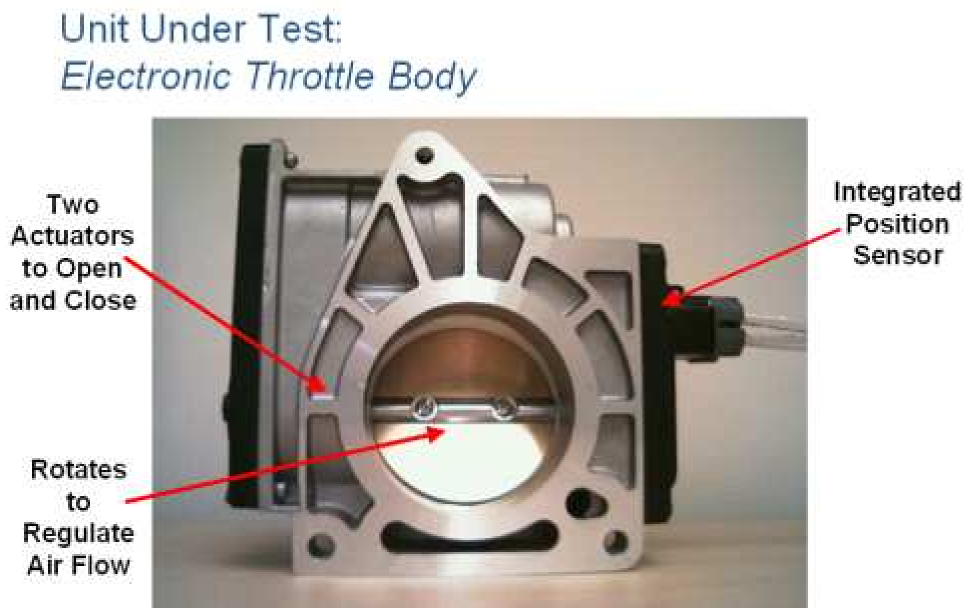
Typographical Convention:

`Fixed Font`	`MATLAB Code.  >> denotes command prompt`
"Do Something"	Double-click a line of text from the example select utility.
Do Something	Follow these directions.
Italics	Example script (what you might want to say).

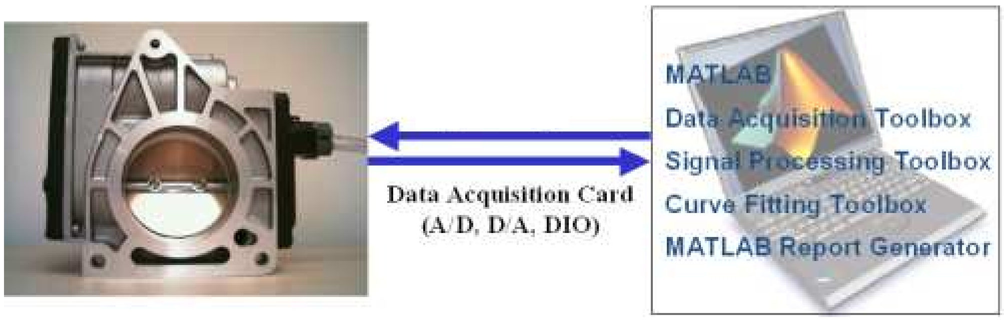
Getting started

<u>Hardware Connections:</u>

NOTE: This demo is for specific hardware used by The MathWorks. It is not recommended that you try to recreate it – just learn from the code!


Unit Under Test:
Electronic Throttle Body
Two Actuators to Open and Close
Rotates to Regulate Air Flow
Integrated Position Sensor

Demo Setup



Software Setup:

This demo requires MATLAB, Data Acquisition Toolbox, Signal Processing Toolbox, and the Curve Fitting Toolbox. Start MATLAB and navigate to the root directory for this case study. This should contain the following files:

demoselector.fig
demoselector.m
demolist.txt

Run the example select utility, which allows you to click through the case study:

```
>> demoselector
```

The path should now be set for the case study. You will be able to access all files by simply double-clicking on the appropriate lines of text in the example selector utility.

Note: The first two sections ("Test and Measurement" and "Data Analysis") you will see in the example selector are use with the Test and Measurement seminar. If you are presenting only this case study, you should scroll down so that "Case Study: Electronic Throttle Body" is visible.

A quick note on the example selector. Here's what you need to know: When you double click on a line of text, MATLAB looks for a dollar sign ($) in the string. It executes all commands listed after the dollar sign. The idea is that the beginning of the string is some useful descriptive text, while the end of the string (containing the commands) is hidden off of the right side of the GUI window. The contents of the example selector come from the example_sel.txt file. The documentation for example_sel.m explains the finer points of using an example_sel.txt file.

Running the Case Study

Note: We will follow the course of action from the case study:

1. **Connect to hardware**
2. **Develop classification algorithm**
   Import data from Excel

Design algorithm
Try out algorithm
Apply to complete data set
Compare with alternative algorithm

3. **Improve simulations with measured data**
   Run throttle simulation
   Improve with measured data
4. **Build integrated measurement system**

1. Connect to hardware

The objective of this step is to connect with the hardware to control the throttle and capture the position response.

"1. Connect to Hardware." (valve_walkthrough.m)
(remember: this means double-click this line)
This opens the file valve_walkthrough.m. Walk through this code one line at a time. Note: We do not show very much code during the case study. This file is written assuming that you have already shown the user the basics of working with the data acquisition toolbox. We have intentionally hidden most of the code details to keep the message clear.

Step 1: Open and close the throttle
Lets start by opening and closing the throttle. I will control it with digital IO – I can set one line high to open it, and a second line high to close it. A spring keeps the throttle slightly open when both lines are low.

```
dio = CreateDIO;

putvalue(dio,[1 0]);            %Open

putvalue(dio,[0 1]);            %Close

putvalue(dio,[0 0]);            %Relax
```

Note: I have configured CreateDIO to look for the Computer Boards PC-CARD-DAS16 card on your computer. If you would like, you can override this feature by specifying the hardware yourself: CreateDIO('mcc',BoardNumber,0:1) (the 0:1 specifies the hardware channels for dio).

Step 2: Record the position response
I'd like to capture the position data directly into MATLAB, so that we can start to analyze it. I will set up an analoginput object to record the position, triggered off of a rising signal

```
%Create and configure ai object
filename = 'LiveData';          %For saving the data
ai = CreateAI(filename);
start(ai)
```

Note: I have configured CreateAI to look for the Computer Boards PC-CARD-DAS16 card on your computer. If you would like, you can override this feature by specifying the hardware yourself: CreateDIO(filename,‘cbi’,BoardNumber,0) (the 0 is the hardware channel). One warning: I have not yet tested this capability!

Note: Be sure to not clear the variable “filename” from the workspace. All of the code you run later looks for this variable to know where it can find the data you record in this section. If it can not find it, the programs will use previously recorded data.

I configured the analog input object to trigger off of a rising signal. When the signal rises above 3 volts (mid-way through the throttle opening), we will capture ¾ second of data. We have used a “pre-trigger”, so that the recording starts ¼ second before *the trigger event.*

```
%Look at the object
ai
```

Notice that we have not acquired any data, since our trigger has not yet occurred.

```
%Open the throttle.  This triggers ai
putvalue(dio,[1 0]);           %Open

ai
```

Now we see that it triggered one time.

```
putvalue(dio,[0 0]);           %Relax
```

Note: be sure to relax the throttle, or you will soon smell smoke as the relays overheat!

Now we will get and plot the data.

```
%Get and plot the data
[time,data] = plotdata(ai);
```

We are looking at the calibrated voltage output from the position sensor. The low value corresponds to the throttle in its resting position. As the throttle opens, this value rises until it reaches it’s maximum value.

Explore the data. Use zoom to look at the two transition points. Turn off zoom, and click on the curve to label data values (see below). Here’s a proposed script (note the shameless plug for my function!):

With MATLAB, it is easy to explore this data. I can interactively zoom in on the regions of interest. I’m interested in determining some precise values from the curve, such as the resting angle or the peak opening angle. There is a user-written function called datalabel that allows you to label points on a curve by clicking on them. This function is available for free download at MATLAB Central.

Note: I have used a function I wrote called datalabel, which will label points on the curve when you click on them. It can be downloaded at MATLAB Central: http://www.mathworks.com/matlabcentral/fileexchange/Files.jsp?fileId=1542

Now we calibrate the data, and export it as a spreadsheet file to share with any unfortunate coworkers who do not have MATLAB.

```
%Calibrate and export as spreadsheet file
export(filename,time,data);
```

The last thing we do is clean up the data acquisition session.

```
stop(ai);
delete([ai dio]);
```

2. Develop classification algorithm

Our goal is to design an analysis system that can tell us if individual throttles are good or bad before we put them into a car.

A. Import data from Excel

We received our data from the test group in an Excel spreadsheet. This is the data from one run. Ignore the fact that we actually already captured data in MATLAB. We would like to show MATLAB's ability to easily import and visualize data.

Import LiveData.xls with Import Wizard (double-click on the file in the Current Directory Browser. It should be in the current directory.) Select "Create vectors from each column using column names"

(Alternate: Select "Import data from Excel" This opens LiveData.xls in import wizard.)

We can get a quick look at our data through a point-and-click interface.

Plot data from workspace browser. You can start to explain our strategy for algorithm development here, or just wait until the next step. The only point of this step was to show that you can quickly get and view data.

B. Design algorithm

I would like to take a closer look at this data, and apply signal processing if necessary.

"Design Algorithm."   This opens sptool

I am using a tool from the Signal Processing Toolbox called sptool. Sptool is designed to provide easy access to common signal processing tasks: data viewing, filter design and implementation, and spectral analysis.

Note: The following list is all in sptool. Key steps are in bold, with details and explanations in the bulleted lists. Make sure that you hit the key steps, but the details are not important. It is more important for you to show off the features of sptool that you think are interesting/useful. My directions assume that you know how to use sptool.

Import data
- From Workspace: position
- Sampling Frequency: fs
- Accept default name of sig1

View data:
- Pan around
- observe slight noisiness
- show off sptool!

The data viewer is useful for exploring time series. It provides advanced zoom and pan features. We can even have it identify all of the local maxima and minima of our data.

Turn on peaks and valleys. We will use these to calculate rise, rise time
- Notice that there are a bunch. Tough to get a good measurement

Note: This is where we explain how we will design our algorithm. The idea is that through exploration, we get some good ideas!

Looking at the signal here gives me some ideas. I think that a "good" throttle will open like this one – it will take about the same amount of time ("rise time") and open through the same angle ("rise"). I would like to automatically calculate these parameters. If I can identify the peak at the end of the opening, and the little valley at the beginning of the opening, I should be OK. The peak calculation does a great job at the end of the opening, but I'm a little worried about it's calculation at the beginning. (Note: Zoom in on these regions as you say this). *Notice how noisy it is; I fear that I won't be able to get consistent measurements. So, I would like to design a low-pass filter to clean up the noise in this data.*

Design a new filter
- We want unity gain in pass-band.
- We won't worry about phase. Since this is offline, we can use zero-phase filtering.
- Zoom into passband. Try a few different filter types. "Stumble" across Butterworth IIR
- Accept default name of filt1. If you use a different name, the subsequent code will use a saved filter, not the one you just designed.

Note: Explain what you are doing, and show how easy it is to try different ideas. Again, just show off sptool!

Apply filter
- Select position and filt1, click Apply
- Select zero phase filtering (filtfilt)
- Accept default name of sig2

Overlay the two traces. Show how filtered one is cleaner
- Select position and position_f in signal list, click View

- Select position as current trace. Show peaks. Select position_f. fewer peaks!
- Deselect position. Now we'll work with smoothed data

Export filt1 (NOTE: don't use TF object!)
Note: close sptool. Don't save the session.

I've developed an algorithm that captures the process we just developed. It looks for the valley just before the signal rises, and the peak just after it finishes rising. I calculate the rise (how much it changes) and the rise time (how long it takes to open). Lets test it out:

C. Try out algorithm

"Try out Algorithm." (analyze_plot.m)
The first figure just shows our raw and filtered data. The bottom plots zoom in on the transition points. Note how we have much smoother data to work with.

The second figure shows the results of our algorithm. All of the peaks and valleys are shown, with the transition points highlighted in black. The computed rise and rise time are also displayed on the figure. Notice how everything is automatic, from the analysis to the results display on the figure.

D. Apply to complete data set

"Apply to Complete Data Set" (rpt_ProductionTest.m)
We will now apply the algorithm to the data from a full durability test. The throttle was run for 100 cycles. Our script applies the filter that we designed, and then runs the smoothed data through the algorithm.

Note: I periodically stuck a pair of plyers into the throttle opening when I recorded the 100 runs. This is how I got some runs to be bad. This data is stored in ValveBatchTest_Relaxed.mat

Our test classifies each cycle as passing or failing, based on the computed rise and rise time. These measurements must be within +/- 5% of the median values. (A look at the data shows why I chose median instead of mean – the distribution is largely one-sided, and I don't want outliers to skew significantly.)

The first figure shows the spread of our data. The rise time and rise angle for every run are shown, along with the acceptable range (the dotted red lines). All bad points are colored black. Again, this is all automatic. A cycle which fails either rise or rise time is classified as failing the test.

Note: datalabels are on, so you can click on a point to identify its run number. More shameless promotion!

You might think we are done, but it is important to look closely at the data, so that we have confidence in our algorithm.

We would like to see more details about the runs, to get a better feeling of if our algorithm is working as expected. Figure 2 shows us the measured position for all of the runs, separated by those that passed and those that failed. The benchmark run (the average of all of the good runs) is overlayed on the failed runs for reference.

Note the one particularly deviant line on the passing run. Can make some comment here about going back to extend the algorithm – the point is that by looking at the data in enough detail, we can identify potential problems. For your info, here's what happened: there was a single dropped sample, which looks like this after filtering (we are effectively seeing the impulse response of the filter you just designed).

There are a couple of more tricks you can do here. You can right-click on a dot in the first figure (the pass-fail results) to highlight the corresponding line in the second figure (the time series). You can also click on lines in the second figure to identify which run they come from. These labels are created with "linelabel" (also at MATLAB Central), which works just like datalabel – click-and-drag to move, right click to delete.

E. Compare with alternative algorithm

There are often many different ways of achieving the same goal. It is difficult to know in advance what will be the best way, so it is important to have flexible tools that encourage and accommodate trying different things. For instance, I decided to focus on identifying the peaks and valleys of our signal in order to compute the rise and rise time. I could just as easily fit a straight line through the rising signal, and look at the slope of the line (rise over rise time). Enough talk – lets give it a try.

We will use the graphical interface provided with the Curve Fitting Toolbox, called cftool. Our goal is to design a fit that will compute the slope of the opening throttle.

Note: My story gets a little bit confusing here. When we present the Test and Measurement Seminar, we introduce the Curve Fitting Toolbox before we get to this part of the case study. We show cftool, and develop the algorithm. At this point in the case study, we just implement the algorithm. My directions here assume that you are presenting the case study without the rest of the seminar. The following is the procedure for designing the algorithm. If you are presenting the entire seminar, you should run this section as an example when you get to the slide on the Curve Fitting Toolbox.

"Curve Fitting". This runs cftool
(This is under Data Analysis, because this is where it is used in the seminar).

I am using a tool provided with the Curve Fitting Toolbox which supports the entire curve fitting process. It gathers all of the core capabilities of the toolbox.

Note: We want to perform a linear fit on just the rising region of data. We will import and then define an exclusion set. I don't provide a script here – just show what you find interesting, and tell them why!
Data:

Data Sets:
- X Data: time
- Y Data: position
- Click apply.

Exclude and Section:
- Graphically or by range
- Exclude values that aren't the rise. Give any name, and click Apply.

Smooth:
- If you like to show smoothing, use Moving Average, Span=5

One comment on the exclude set: When we exclude graphically, we end up defining which samples we will exclude; i.e. the first 100 and the last 300. This could get you in trouble if your data is not consistent. It will be OK for us, because of the way we trigger our data. An alternative would be to exclude by value – leave out all data that's below 25 and above 85.

Fit:
- use smooth data set, exclude region
- Linear Fit
- Slope (rise / rise_time) is p1 in Results section

Note: You may want to show that there are many more options for fitting, allowing us to have as much detail as we like.

Main Window:
- Explore data some more
- data tips
- View: Prediction Bounds
- Tools: Axis Limit Control

Now I have developed an algorithm that I like. I would like to apply it to hundreds of data sets, but I don't want to spend all day clicking. Fortunately, cftool can write an m-file for me which will allow us to apply this fit to any similar data.

- Save M-file as throttle_cfit_new

Run the file that was created, to show what it does. If you like, run it with some other data, too.

```
>> throttle_cfit_new(time,position);
```

Since this is just MATLAB code, we can easily go in and make changes. We modified the generated m-file to do the following things:
- *Returns the slope of the fitted line*
- *Displays slope on the plot*
- *Has an option to not graph results*

We can now run our same production test, but using the new analysis. We have a single parameter (slope), which is simple the ratio of the two parameters from our previously developed analysis. We again define an acceptable range or +/-5% of the median value, and will generate the same figures

(Note: Back to the case study!).
"Compare with alternative algorithm" (cfit_analysis.m)

This figure shows the results of this algorithm along with the results of the previous (signal processing-based) algorithm. We have only a single variable – the slope of measured position. The previous algorithm gave us rise and rise time; the slope is just the ratio.

This gives about the same results as our previous analysis – it is up to us to decide which is better. Notice that having flexible graphics allows us to put the information in the right perspective so that we can make informed decisions.

3. Improve simulations with measured data

Simulation is another major aspect of component and system-level design. We have developed a Simulink model of the throttle body for incorporation into a larger system-level simulation. Since this model is developed in the same environment as our analysis, it is easy to compare the results. This allows us to tune our simulation to accurately reflect our real system. Additonally, we can leverage the analysis and visualization routines we have already developed for testing our simulation.

A. Run original throttle simulation

We start by running the Simulink model. We have modeled the throttle body as a damped, driven harmonic oscillator with nonlinearities. As we've seen from the measured data, there is a hard stop when the throttle swings all of the way open.

"Run throttle simulation" throttle_simulation.mdl

Explain enough about the model so that people will understand, and relate it to the real world: we have a step input, the model of the throttle dynamics, and the measured position response is output. When you run the model, you can see the data in the scope. Another figure will open, which compares the measured and simulated data. It also runs the simulated data through the algorithm we developed earlier.

I'd like to point out a few interesting things from this plot. First, you might notice that my results are pretty good, but not great – I'll get to that in a minute. Second, is the fact that I could easily create this figure comparing my simulation and measured data. Notice that I ran my simulated response through the same algorithm I developed for my measured data. Also, notice that my simulation runs over the same time basis, and at the same sample rate, as my measured data. Since my simulation "knows" how my data was recorded, this is easy (and automatic). Think about making a comparison like this when using different software packages for simulation, test, and data analysis.

Back to the results. Notice the slow rise time, and also that the limits are off (it starts around 18 degrees and peaks at 90 degrees). How can we improve on this?

B. Improve with measured data

A simple first step would be to use the measured limits – the starting position, the ending position, and the peak opening (hard stop). We will also fine tune the model parameters. We could use a whole host of MATLAB's tricks - curve fitting, optimization, etc. We will just use a simple parameter sweep. We will change the stiffness coefficient in small increments and plot the results. Since we can can control Simulink from MATLAB, it is easy to run batch simulations like this. The figure compares the simulation output from each run with the measured data. The best match (defined as minimum least square error between the measured and simulated response) is highlighted in yellow.

"Improve with measured data" throttle_simulation_validate.m

Note: People have generally been really impressed at how well we can get the simulation to match the measured data.

Note: I do one additional step to compare the measured and simulated responses. I shift the simulation response so that it matches the measured response at the trigger. The measured response was triggered at 3V (60 degrees). This trigger is defined as t=0. I find the time at which the simulation response equals this trigger value, and shift the response so that this occurs at simulated time=0. I'm not sure if you'd want to explain this.

There are many advantages of having simulation informed by test:
- We can use the same algorithms and analysis for our measured and simulated data.
- Our simulation runs at the same time steps as our measured data, making it very easy to compare measured and simulated data
- We can immediately and automatically improve our simulations with measured data.

4. Build integrated measurement system

Now we are ready to tie everything together. We will integrate the test component we developed earlier with the analysis we just developed. We have built an application which runs the complete test, performs the data analysis, and generates a report of the results. The application is designed to allow somebody with no MATLAB knowledge to run the test. The report documents the development and testing of our algorithm along with the test results. To play it safe, we will also include all of the original data and the transition points computed by the algorithm. We also document the simulation and the compared results.

Select: "Create Report." Click "Run"

"4. Build integrated measurement system" (valve_gui_batch.m)

Also notice that we give the operator live feedback during the test by displaying not only the measured data, but the output of our analysis. By performing the post-test analysis *on the fly, we can make better use of our testing time by aborting failing tests or altering our test in light of our analysis (e.g.).*

Note: You might need to go into the file valvebatch_function to configure for your hardware. It should automatically find the Computer Boards card, but I haven't had the chance to test it!

Appendix A: File List. Organized by Presentation Order

T&M Section Walkthrough:

valve_walkthrough.m	Everything you need for walkthrough. Step through manually

Files called directly by valve_walkthrough:

createDIO.m	Create and configure DIO Object
createAI.m	Create and configure AI object.
ConfigureTrigger.m	Configures the ai trigger
plotdata.m	Get, plot, and calibrate one trigger of data
export.m	Export the data to a Microsoft Excel file.

Analysis Section Walkthrough:

These are accessed directly through the example select utility.

analyze_plot.m	Apply algorithm and plot results
rpt_ProductionTest.m	Apply to a large set of data. Funny name, because it is used for other stuff
cfit_analysis.m	Try using curve fitting to design the algorithm.
throttle_cfit.m	Customized cftool m-file; implements algorithm
throttle_simulation.mdl	Simulink model of the throttle. Initial parameters are intentionally not very good.
throttle_simulation_validate.m	Code to walkthrough validation/tuning of simulation.
valve_gui_batch.m,.fig	GUI to run complete test, process results, and generate report. Aka "the whole shebang."

Supporting Files:

find_transition.m	THE algorithm. Finds transition points for position response
ValveBatchTest_Relaxed.daq	A good data set. Used in throttle_simulation_validate.
ValveTestRelaxed.mat	.mat file containing one recorded run. Often used if no recently recorded data is available.
ValveTestRelaxed.xls	Excel file containing one recorded run (same as ValveTestRelaxed.xls). Use to show import wizard.
ValveTest_Relaxed.mat	-MAT file version of the Excel file. Used for safety..
valve_sptool_export.mat	The lowpass filter we designed with sptool in the analysis section
throttle_simulation_stopfcn.m	StopFcn for throttle_simulation. Used for parameter sweep only. Shows simulation output for a wide range of stiffness coefficients.

throttle_simulation_stopfcn1run.m	StopFcn for throttle_simulation. Used for single runs.
throttle_simulation_init.m	Initialization routine for throttle_simulation. Defines parameter values.
Valve_Analysis.rpt	Report Generator setup file. Used for both online and offline analysis
rpt_AlgorithmDesign.m	Script for Chapter 1 of report. Used for online and offline
rpt_ProductionTest.m	Script for Chapter 2 of report. Used for online and offline
rpt_AppendixDetails.m	Script for Chapter 3 of report. Used for online and offline
demolist.txt	Configures example selection utility to provide access to the demo files for this case study.

Utilities

subplot_varspace.m	Custom subplot lets you specify spacing between axes
daqtriggerreshape.m	Reshapes a NaN separated vector of triggered data into a matrix; one column per trigger
datalabel.m	Interactive labeling of data points
suptitle.m	Add a single title above multiple subplots
fullscreen.m	Make a figure fill the screen
findpeaks.m	Find local maxima of a function. Extracted from sptool
linelabel.m	Interactive labeling of lines.
demoselector.m	Tool to easily find and run demos.
xlswrite.m	Write a basic Microsoft Excel spreadsheet.

Documentation

Throttle Case Study.doc	This document.

Additional Files for Seminar
"MATLAB for Test Data Analysis"

The demos for this section are divided into 2 groups: 1) simple examples to show customers how to work with the toolboxes, 2) complete examples to show off functionality / capability of the toolboxes.

1) Simple Examples
All files live in the "Quick Product Demos" directory.
The most useful ones can be accessed from the example selector, under the "Test and Measurement" heading

daq_session.m	Code to walk through the minimal daq session
daq1.m	Simple analog output example
instr_session.m	Code to walk through the minimal instrument ctrl session
instr1.m	Simple oscilloscope communication example
getscopedata_gui.m, .fig	Simple GUI to upload data from oscilloscope.

Appendix B: File List. Organized by Directory

All of the Important Stuff

analyze_plot.m	Apply algorithm and plot results
export.m	Export the data to a Microsoft Excel file.
plotdata.m	Get, plot, and calibrate one trigger of data
rpt_AlgorithmDesign.m	Script for Chapter 1 of report. Used for online and offline
rpt_AppendixDetails.m	Script for Chapter 3 of report. Used for online and offline
rpt_ProductionTest.m	Script for Chapter 2 of report. Used for online and offline
valve_Analysis.rpt	Report Generator setup file. Used for both online and offline analysis
valve_walkthrough.m	Everything you need for T&M walkthrough. Step through manually.

Curve Fit Exploration

cfit_analysis.m	Try using curve fitting to design the algorithm.
throttle_cfit.m	Customized cftool m-file; implements algorithm
throttle_cfit.cfit	cftool session file used to develop throttle_cfit.m

Data Files

ValveBatchTest_Relaxed.daq	A good data set. Used in throttle_simulation_validate.
ValveBatchTest_Relaxed.mat	-MAT version of same file.
valve_sptool_export.mat	The lowpass filter we designed with sptool in the analysis section
ValveTestRelaxed.mat	.mat file containing one recorded run. Often used if no recently recorded data is available.
ValveTestRelaxed.xls	Excel file containing one recorded run (same as ValveTestRelaxed.xls). Use to show import wizard.
valve_sptool.spt	sptool session file for the case study.
valve_sptool_export.mat	The data you would export from sptool for the case study.

Documentation

Throttle Case Study.doc	This document.

GUIs

valve_gui_batch.m,.fig	GUI to run complete test, process results, and generate report. Aka "the whole shebang."
valvebatch_function.m	This function is used by valve_gui_batch to actually run the hardware. It is a separate file because it can be used to run tests without the GUI.
valve_gui_oc.m,.fig	A GUI to control the throttle and view it's response
valve_gui_simple.m,.fig	A GUI to just control the throttle (open/close/relax)

Measurement Support

CreateDIO.m	Create and configure DIO Object
CreateAI.m	Create and configure AI object.

ConfigureTrigger.m	Configures the ai trigger
Quick Product Demos	
comparedemo_stats	A simple stats demo. Not used.
datainsight_signal.m	A simple signal processing demo. Used to highlight Signal Processing Toolbox during the seminar
daq_session.m	Code to walk through the minimal daq session
daq1.m	Simple analog output example
getscopedata_gui.m, .fig	Simple GUI to upload data from oscilloscope.
instr_session.m	Code to walk through the minimal instrument ctrl session
instr1.m	Simple oscilloscope communication example
instr1_scope.m	More compact oscilloscope example
Simulink	
throttle_simulation.mdl	Simulink model of the throttle. Initial parameters are intentionally not very good.
throttle_simulation_init.m	Initialization routine for throttle_simulation. Defines parameter values.
throttle_simulation_stopfcn.m	StopFcn for throttle_simulation. Used for parameter sweep only. Shows simulation output for a wide range of stiffness coefficients.
throttle_simulation_stopfcn1run.m	StopFcn for throttle_simulation. Used for single runs.
throttle_simulation_validate.m	Code to walkthrough validation/tuning of simulation.
utilities	
daqtriggerreshape.m	Reshapes a NaN separated vector of triggered data into a matrix; one column per trigger
datalabel.m	Interactive labeling of data points
demoselector.m,.fig	Tool to easily find and run demos.
findpeaks.m	Find local maxima of a function. Extracted from sptool
fullscreen.m	Make a figure fill the screen
linelabel.m	Interactive labeling of lines.
subplot_varspace.m	Custom subplot lets you specify spacing between axes
suptitle.m	Add a single title above multiple subplots
xlswrite.m	Write a basic Microsoft Excel spreadsheet.